پروژه‌ی پایانی ساختمان داده

نیمسال اول 1402

استاد الهام افشار

دانشجویان:

آریا نادری 40112358043

رومینا خانمحمدی 40112358012

ترانه بهمنی 9912358010

فهرست:

مقدمه

بنابر توضیحات داده شده در داخل سند پروژه ی سمنحش، باید سامانه ای طراحی و پیاده سازی کنیم که منجر به محاسبه و نمایش بهترین مسیرها بر اساس زمان، مسافت و هزینه براساس مبدا و مقصدی ورودی توسط کاربر شود.

روش کار برنامه ی ما به این صورت است که ابتدا تعداد test case را از کاربر دریافت و سپس برای هرکدام از ورودی ها مبدا و مقصد را به توابع مربوطه می فرستیم، بعد از آن برنامه اجرا می شود.

در مرحله ی نخست کمترین مسافت طی شده محاسبه و همراه آن، مسیری که باید کاربر طی کند، وسیله ی نقلیه ای که باید ازآن استفاده شود و همچنین مدت زمانی که طول می‌کشد تا کاربر این مسیر را با وسیله های نقلیه ی گفته شده طی کند، چاپ می‌شود.

در مرحله ی بعدی کمترین هزینه ی ممکن برای طی کردن مسافت بین مبدا و مقصد داده شده را چاپ و همزمان مسیر مورد نظر کاربر به همراه وسیله ی نقلیه ای که باید استفاده شود و مدت زمانی که طول می‌کشد کاربر با شرایط فوق از مبدا به مقصد برسد، را نمایش می‌دهد.

درمرحله ی آخر نیز کمترین زمان ممکن به همراه مسیری که کاربر دراین زمان باید طی کند را چاپ می کند.

جزییات بیشتر برنامه و کلاس های موجود درآن را می‌توانید در ادامه ی گزارش مشاهده کنید.

کلاس Vehicle:

این کلاس شامل سه Data member می‌باشد که در آن مقدار مدنظر (value)، لاین و نوع وسیله نقلیه ذخیره می‌شود.

همانطور که در نمای کد پایین مشاهده می‌شود، درconstructor این کلاس مقدار distance صفر و نوع وسیله ی نقلیه و لاین نیز با یک string خالی پر می‌شود.

توابع get-val و get-type-vehicle به ترتیب نوع وسیله ی نقلیه و value را return می‌کنند.

```
class vehicle
{
    private:
    unsigned int value;
    string line_vic;
    string vic_type;

    public:
    unsigned int get_val(){return value;}
    void setval(int val);
    void set_line(string line);
    void set_vic(string vic);
    vehicle(){value = 0; line_vic = ""; vic_type = "" ;}
    string get_line(){return line_vic;}
    string get_vic(){return vic_type;}
```

توابع setval و setvic به ترتیب value و نوع وسیله‌ی نقلیه را set می‌کنند.

کلاس Station:

در این بخش ما سه Data member از جنس کلاس Vehicle داریم (که نوع وسیله نقلیه‌ی ما را مشخص می‌کنند)

تابع setinfo برای set کردن مقادیر مدنظر استفاده شده است.

تابع getdis وظیفه‌ی return وسیله‌ی نقلیه ایست که کمترین مقدار(برای مثال کمترین مسافت) و در حالت دیگر overload شده برای return وسیله‌ی نقلیه مدنظر استفاده می‌شود.

تابع get_time وظیفه‌ی محاسبه‌ی مدت زمان جابه‌جایی را با توجه به ساعت شروع و درنظر گرفتن ساعت ترافیک دارد.

```
class station
{
    private:
    vehicle bus;
    vehicle metro;
    vehicle taxi;


    public:       class vehicle
    void setinfo(vehicle value); //set value
    vehicle getdis(); // return vehicle that have minimum value
    vehicle getdis(const string & type_vehicle); // overload getdis that return appropriate vehicle
    station(){};
    int get_time(string vehicle, bool flag, Time start_time); // return time of appropriate vehicle
    Time get_time(int time , Time start_time);

};
```

کلاس Time:

این کلاس شامل دو Data member ساعت و دقیقه می باشد (که در ابتدا با صفر مقداردهی شده اند).

در این بخش ما operator + را overload کرده ایم که به کمک آن یک شی از کلاس تایم می‌توانند با یک عدد int (به عنوان دقیقه‌ی حساب شده) جمع شود.

توابع get-hour و get-minute نیز به ساعت و دقیقه را return می‌کنند.

تابع print هم وظیفه‌ی چاپ کردن زمان به فرم استاندارد (مثلا 10:27) را دارد.

```
class Time
{
    private:
    int hour = 0;
    int min = 0;

    public:
    Time(const std::string & start_time);
    void operator+(int minute); // operator overloading
    void print();
    int get_hour(){return hour;}
    int get_minute(){return min;}
};
```

کلاس smnhsh (کلاس اصلی):

اصلی ترین کلاس این برنامه است که توابع اصلی برنامه در این کلاس تعریف شده اند.

این کلاس دارای یک constructor است که درآن دو تابع `read_distance_from_file` و `complete_graph_for_cost` فراخوانی می‌شوند.

در تابع read-distance-from-file به ترتیب اسم ایستگاه ها به همراه مسافت آن ها از فایل خوانده شده این و اطلاعات در مپ مربوطه ذخیره می‌شوند.

پرشدن ماتریس مسافت، مشخص شدن لاین ها و همچنین اندیس گذاری ایستگاه ها که بعدا در ماتریس مجاورت برای پیداکردن ایستگاه نیازداریم، در همین تابع انجام می‌شود.

**ماتریس مجاورت[1]: یک ماتریس 59 * 59 است که نشان دهنده راه های ارتباطی بین ایستگاه ها است.

تابع دیگر تابع complete_graph_for_cost است که برای محاسبه بهترین هزینه، تمامی ایستگاه هایی که در یک لاین مترو یا تاکسی هستند را به یکدیگر متصل می‌کند.

Data member های این کلاس شامل دو ماتریس path و cost بوده که همانطور که از اسم آنها مشخص است ماتریس اولی برای مسافت و ماتریس دومی برای هزینه است. هر دوی این ماتریس هایpath

[1] Adjacency matrix

و cost از نوع کلاس station هستند که هر شیء ازکلاس station هم در داخل خودش دارای 3 Data member دیگر از کلاس vehicle می‌باشد.

همچنین در این کلاس از سه unordered_map[2] ابه نام های name_of_station (برای اتصال نام هر ایستگاه به یک اندیس) ، lines (برای اتصال هر لاین به ایستگاه ها مربوطه) و station_vehicle (که نام هر ایستگاه را در ابتدا به لاین مربوطه و سپس وسیله‌ی نقلیه‌ی آن متصل می‌کند) استفاده شده است.

در تصویر زیر یک نمای کلی از این کلاس و توابع آن را مشاهده می‌کنید.

```
class smnhsh
{
    private:
    station pathes [59][59] = {}; //this matrix store every information between each two node i and j like line, vehicle and distance
    station costs  [59][59] = {}; // this matrix store cost between two node
    unordered_map <string, int > names_of_station; // links name of every station to a index
    unordered_map <string, vector<string>> lines; //for linking each line to its stations
    unordered_map <string, unordered_map<string, unordered_set<string>>> station_vechicle; // link name of every station to its lines
                                                                                           // then link each line to its avalible vehicle

    public:
    smnhsh(); //constructor
    bool is_valid(const string & start, const string & end) const; //checking name of stations(does exist or not)
    void run(); //run program
    void get_input(); //get inputs from users
    void read_distance_from_file (); //reading information from file
    void complete_graph_for_cost();  //make a graph filled with costs
    string search_in_map(int);  // it used for find name of a station with a number with search in map
    int  minvalue(const node dist[], const bool sptSet[]) const; // it used for dijkestra
    void find_short_path(const int & start, const int & end , Time & start_time); // it is dijkestra
    void find_lowest_cost(const int & start, const int & end, Time & start_time); // it is dijkestra
    void show_shortest_path(const node & pathe, Time start_time); //print direction for shortest path and calculate arriving time
    void show_cost(const vector <string> & , const vector <string> & , const vector <string> & , Time);//print directrion for lowest cost
    void calculate_time_each_line(unordered_map<string, unordered_set<string>> , string , node array[], bool visible[], Time &start_time);
    void find_lowest_time(const int &start, const int &end, Time &start_time);
    void print_lowest_time(const node &path, Time start_time); // print direction for lowest time
```

[2] https://www.geeksforgeeks.org/unordered_map-in-cpp-stl

تابع **is-valid**: برای بررسی معتبر بودن ورودی های وارد شده (برای مثال صحیح بودن نام ایستگاه های دریافتی) توسط کاربر استفاده می شود.

تابع **run**: این تابع در main.cpp فراخوانی می‌شود که در آن مجددا تابعی دیگر به نام get_input() فراخوانی می شود .

تابع **get-input**: ورودی ها را که شامل مبدا و مقصد و زمان شروع حرکت است را از کاربر دریافت کرده و اطلاعات را به توابع مربوطه (find_shortest_path , find_lowest_cost, find_lowest_time) ارسال می‌کند.

تابع **complete-graph-for-cost**: در این تابع ماتریس costs براساس هزینه ی بین هر دو ایستگاه پر می‌شود؛ به این شیوه که در ابتدا در این ماتریس تمامی ایستگاه هایی که از طریق لاین اتوبوس به هم متصل هستند (چون مبلغ جابه‌جایی با اتوبوس کمتر است) را مشخص کرده و هزینه‌ی مربوطه را برای آن ها ذخیره می کند. پس از آن نیز در صورتی که قیمتی برای جابه‌جایی بین دو ایستگاه تعیین نشده باشد و یا اینکه هزینه‌ی جدیدکمتر از مبلغ قبلی باشد ماتریس را پر می‌کند.

تابع search-in-map: برای یافتن ایستگاه مورد نظر براساس اندیس نسبت داده شده به آن استفاده می شود (در واقع از آنجایی که دسترسی دو طرفه بین اسم ایستگاه ها و اندیس آنها وجود ندارد از این تابع کمک می‌گیریم که اسم ایستگاه را بر اساس اندیس آن پیدا کنیم.)

تابع minvalue: این تابع کمترین مقدار مورد نظر را (برای بخش دایجسترا) return می‌کند.

تابع find_shortest_path : این تابع با دریافت مبدا، مقصد و زمان شروع حرکت به عنوان ورودی، به وسیله‌ی الگوریتم دایجسترا، کمترین مسافت بین دو ایستگاه را یافته و برای چاپ آن مسیر تابع show-shortest-path را فراخوانی می‌کند.

تابع show-shortest-path: این تابع مقدار کمترین مسافت طی شده، مسیر آن و همچنین زمان لازم برای طی کردن این مسیر(arriving time) را برای ما چاپ می‌کند.

تابع find_lowest _cost : این تابع نیز با دریافت مبدا، مقصد و زمان شروع حرکت به عنوان ورودی، به وسیله‌ی الگوریتم دایجسترا و به وسیله‌ی ماتریس costs کمترین مسافت بین دو ایستگاه را یافته و برای چاپ آن مسیر تابع show-shortest-path را فراخوانی می‌کند.

تابع show-cost : این تابع مقدار کمترین هزینه، مسیر آن و همچنین زمان لازم برای طی کردن این مسیر(arriving time) را برای ما چاپ می‌کند .

تابع find_lowest _time : این تابع نیز مانند توابع بالا با دریافت مبدا، مقصد و زمان شروع حرکت به عنوان ورودی، به وسیله‌ی الگوریتم دایجسترا (کمی متفاوت تر از قبل) کمترین زمان را یافته و با فراخوانی تابع calculate_time_each_line عملیات محاسبه‌ی کمترین زمان را تکمیل می‌کند.

تابع calculate_time_each_line : این تابع برای محاسبه‌ی کمترین زمان ممکن طراحی شده که در آن با بررسی شرط های لازم و مورد نیاز (برای فعال کردن یک flag در مراحل مختلف)

پس از طی کردن تمام حالات موجود و محاسبه یه ساعت حرکت اعمال تغییرات زمانی بر آن می‌تواند کمترین زمان ممکن را مشخص کند.

تابع print_lowest_time: این تابع مقدار کمترین و بهترین زمان ممکن، مسیر طی شده به همراه وسیله‌ی نقلیه مورد استفاده قرار داده شده را چاپ می‌کند.

چالش ها و الگوریتم:

در این پروژه هدف پیدا کردن کمترین مسافت، کمترین زمان و کمترین تایم ممکن روی نقشه‌ی (گراف) تهران است. بین هر دو ایستگاه مقادیر متفاوتی مانند جمله مسافت، زمان، و هزینه وجود دارد. در هر سه مرحله مساله یافتن مسیری با کم وزن ترین یال های ممکن از مبدا به مقصد است. برای پیدا کردن کمترین مسافت، دقیقا از الگوریتم دایجسترا[3] استفاده شده. از چالش های این قسمت میتوان به شیوه ذخیره سازی اطلاعات و مسافت ها به گونه ای که قابلیت استفاده مجدد (reusable) و محاسبه‌ی کمترین زمان طی شده با توجه به نیاز برای بررسی کردن تمام مسیر ها و حالت های ممکن که در زمان های مختلف (مخصوصا تایم ترافیک) اشاره کرد. برای محاسبه‌ی کمترین هزینه استفاده از الگوریتم دایجسترا به صورت پیش فرض، به دلیل خط به خط محسابه شدن هزینه مترو و اتوبوس ممکن نبود؛ این مشکل با وصل کردن تمامی ایستگاه هایی که در یک خط مترو یا اتوبوس قرار داشتند به دیگر ایستگاه های همان خط (به نوعی کامل کردن گراف) برطرف شد.

در قسمت زمان نیز استفاده از دایجسترا به صورت پیش فرض ممکن نبود. الگوریتم مورد استفاده در این قسمت به این صورت است که از یک ایستگاه با تمامی وسیله های نقلیه موجود به کل ایستگاه های همان خط طی می‌شد و زمان طی شده در هر مرحله ذخیره می‌شد. در مرحله بعدی با چک کردن وسیله های دیگر و یا ایستگاه های دیگر، اگر زمان کمتری محسابه شود جایگزین میشود و بدین ترتیب تمامی

[3] Dijkstra algorithm

ایستگاه های هر خط با تمامی وسیله های ممکن چک می‌شوند و در نهایت کمترین هزینه به تمامی ایستگاه ها به دست می‌آید.

فایل‌ها:

تنها فایل موجود در پروژه، فایل تکست taxi-bus-distance است که در آن شماره لاین‌ها، نام ایستگاه ها به همراه مسافت بین آن ها ذخیره شده است.

برای مشخص کردن انتهای هر لاین کلمه ی End در فایل نوشته شده و نشان دهنده‌ی شروع لاین است.

بخش گرافیکی:

برای بخش گرافیکی از qml استفاده شده که یک زبان برای طراحی رابط گرافیکی بر پایه JavaScript است که در چهارچوب Qt مورد استفاده قرار می‌گیرد.

در این قسمت برای ایجاد node ها و لاین ها از مستطیل (Rectangle) استفاده شده که ابعاد آن با توجه به نیاز بخش موردنظر تعیین شده است.

در qml هر شی می‌تواند دارای ویژگی های خاص باشد(که به آن id می‌گویند).

برای هر ایستگاه (هر node) یک id منحصر به فرد به کار رفته که به وسیله‌ی آن امکان ارجاع اشیا به یکدیگر فراهم می‌شود. در واقع می‌توان موقعیت مکانی اشیا دیگر (در این پروژه node ها و گاها لاین ها) را بر اساس سایر اشیا و به کمک id مشخص کرد.

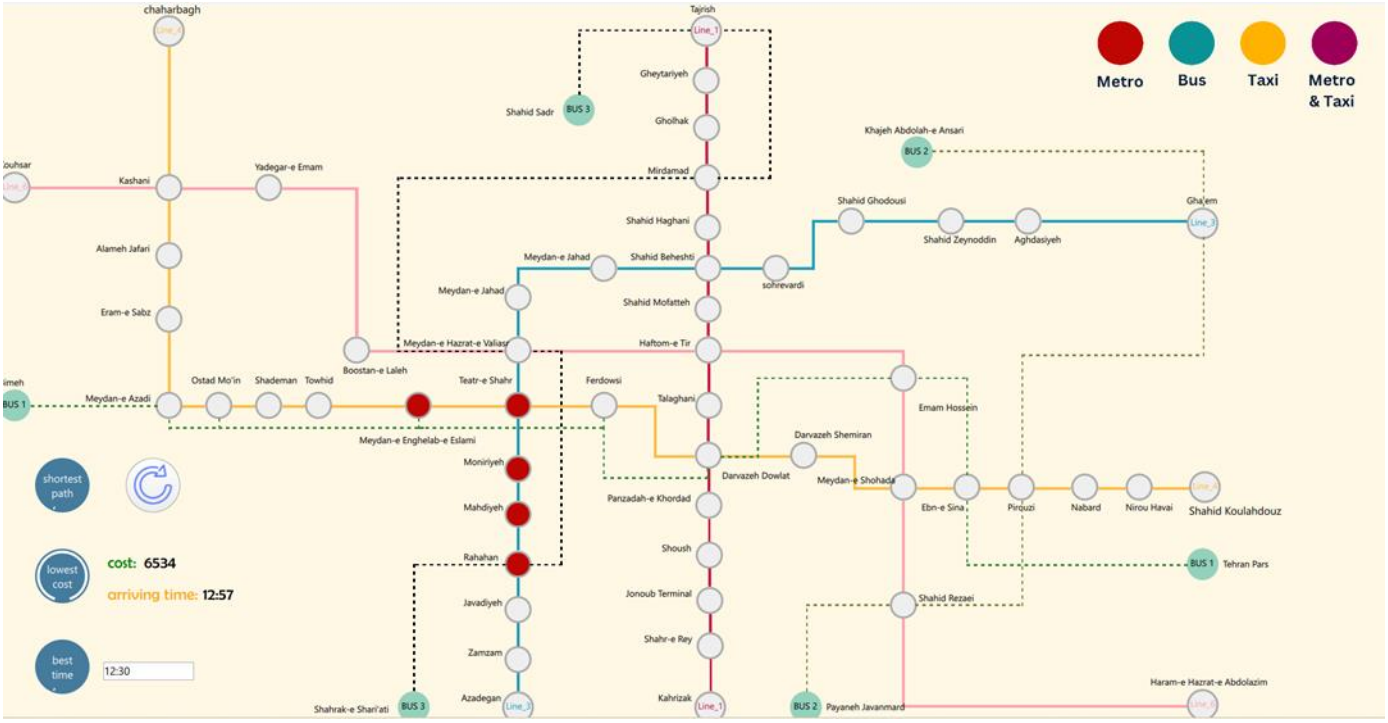
در این قسمت کاربر پس از مشخص کردن مبدا و مقصد موردنظر می تواند با انتخاب یکی از سه دکمه‌ی shortest path , lowest cost و best time مسیر مورد نظر برای این حالت را بر روی نقشه مشاهده کند.

همچنین برای تنظیم مجدد باید از دکمه‌ی reset استفاده شود .

برای کار با برنامه کاربر موظف است که ساعت ورودی را وارد کند (در غیر این صورت برنامه اجرا نمی‌شود)

پس از وارد شدن تمامی اطلاعات ماربر می تواند arriving time و یا مقدار مد نظر خود را (مسافت یا هزینه یا زمان) را مشاهده کند.

نمایی از برنامه:



لینک‌ها:

این پروژه در مخرن گیت‌هاب زیر قرار گرفته است:

https://github.com/arya237/DS.git

منابع:

https://www.geeksforgeeks.org

https://en.cppreference.com